International Journal on Human Machine Interaction [IJHMI]

ISSN	2356-556X		Vol& Issue	V1.I1 @ Jan 2014
Website	ijhmi.asdfjournals.com		eMail	eic.ijhmi@asdfjournals.com
Received	13-Nov-2013		Accepted	20-Dec-2013
Article ID	IJHMI2014005		eAID	e-aid.org/IJHMI.2014.005

Real-Time Task for Embedded Control Systems

Atef Gharbi[1], Mohamed Khalgui[2], Samir Ben Ahmed[3]

[1]LISI, INSAT, Tunis, Tunisia
[2]ITIA, CNR Institute, Italy
[3]FST, University of El Manar, Tunis, Tunisia

Abstract- *This paper deals with the study of Real-Time Tasks for Embedded Control Systems following component-based technology. A real-time task is assumed to be a set of components having some properties independently from any real-time operating system. We apply the Priority Ceiling Protocol (PCP) as a method to ensure the scheduling between periodic tasks with precedence and mutual exclusion constraints. We simulate the scheduling of Real-time tasks with PCP through the Cheddar tool.*

Keywords- Real-Time Tasks; Embedded Control System; Simulation; Priority Ceiling Protocol (PCP)

I. INTRODUCTION

Nowadays, the time to market the embedded systems is becoming shorter and the cost of development is being cheaper. In order to realize this goal (i.e. reducing time and cost), the designers of embedded system propose to reuse the already developed software components. The main benefits of component are: (i) the separation of concerns which means that each component can be defined alone after that all the system can be reproduced; (ii) the modification of component is very simple (such as modification of data, algorithm, connection, …);(iii) the adaptation of component to any modification; (iv) there use of component (it is not specific for only one application).

However, there are few kinds of component-based technologies (such as Koala [1], PBO [2], PECOS [3], …) used in the development of embedded control system due to extra-functional properties to be verified (for example quality of service, timeliness, …) [4]. Anyway, each component-based technology has its benefits and its drawbacks.

Although component technologies deal successfully with functional attributes, they provide no support for managing extra-functional properties of systems (such as synchronization, memory optimization, power consumption, and temporal attributes) that cannot be encapsulated in one component with well-defined interfaces as they crosscut the structure of the overall system. To do so, we define at the operational level some sequential program units called real-time tasks. Thus, we define a real-time task as a set of components having some real-time constraints. We characterize a task by a set of properties independently from any Real Time Operating System (RTOS).

We study in particular the scheduling of tasks through a Real Time Operating System. We apply the priority ceiling protocol proposed by [5] to avoid the problem of priority inversion as well as the deadlock between the different tasks. The priority ceiling protocol supposes that each semaphore is assigned a priority ceiling which is equal to the highest priority task using this semaphore. Any task is only allowed to enter its critical section if its assigned priority is higher than the priority ceilings of all semaphores currently locked by other tasks. The contributions of this research work have been applied to the benchmarking production system: FESTO system (used as running example) allowing us to validate our results.

The Section 2 presents the benchmark production system (FESTO system). We define in Section 3 the scheduling with the Priority Ceiling Protocol. The Section 4 treats the simulation of real-time tasks with the Cheddar tool. Finally, we conclude in the last section.

II. BENCHMARK PRODUCTION SYSTEM

We present the Benchmark Production System (Detailed descriptions are available in the website: http://aut.informatik.uni-halle.de): FESTO available in the research laboratory at the Martin Luther University in Germany. The FESTO Benchmark Production System is a well-documented demonstrator used by many universities for research and education purposes, and it is used as a running example in the context of this paper. FESTO is composed of three units: Distribution, Test and Processing units (Figure 1). The Distribution unit is composed of a pneumatic feeder and a converter to forward cylindrical work pieces from a stack to the testing unit which is composed of the detector, the tester and the elevator. This unit performs checks on work pieces for height, material type and color. Work pieces that successfully pass this check are forwarded to the rotating disk of the Processing unit, where the drilling of the work piece is performed. We assume in this research work two drilling machines Drill_machine1 and Drill_machine2 to drill pieces. The result of the drilling operation is next checked by the checking machine and the work piece is forwarded to another mechanical unit.

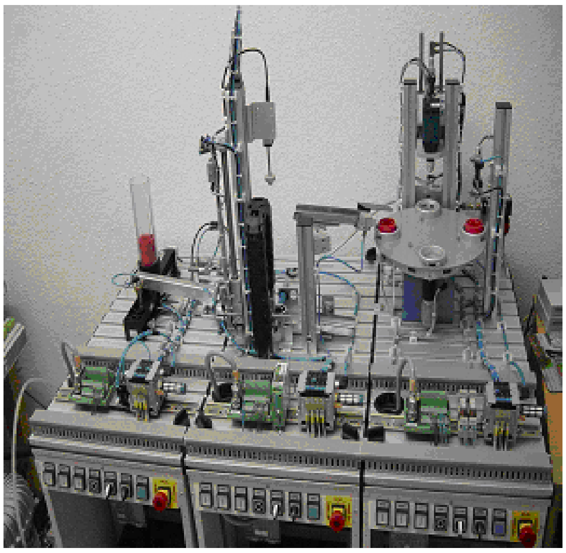


Figure 1. The FESTO system

III. TASK SCHEDULING WITH PRIORITY CEILING PROTOCOL

How to schedule periodic tasks with precedence and mutual exclusion constraints is considered
as important as how to represent a task in a general real-time operating system. In our context, we choose the priority-driven preemptive scheduling used in the most real-time operating systems. The semaphore solution can lead to the problem of priority inversion which consists that a high priority task can be blocked by a lower priority task. To avoid such problem, we propose to apply the priority inheritance protocol proposed by (Sha, 1990).

The priority inheritance protocol can be used to schedule a set of periodic tasks having exclusive access to common resources protected by semaphores. To do so, each semaphore is assigned a priority ceiling which is equal to the highest priority task using this semaphore. A task Γ_i is allowed to enter its critical section only if its assigned priority is higher than the priority ceilings of all semaphores currently locked by tasks other than Γ_i.

Schedulability test for the priority ceiling protocol: a set of n periodic tasks using the priority ceiling protocol can be scheduled by the rate-monotonic algorithm if the following inequalities hold,

$$\forall\, i,\ 1 \le i \le n,\ \ C_1/T_1 + C_2/T_2 + \ldots + C_i/T_i + B_i/T_i \ \le\ i(2^{1/i}-1)$$

where B_i denotes the worst-case blocking time of a task Γ_i by lower priority tasks.

Running Example. In the FESTO Benchmark Production System, we consider three tasks *R1* (a reconfiguration task), *S1* and *S2* (service tasks) having as priority *p1*, *p2* and *p3* such that *p1>p2>p3*.

The sequence of processing steps for each task is as defined in the section previous paragraph where S (resp. R) denotes the service (resp. reconfiguration) semaphore:
R1 = { … P(R) execute reconfiguration V(R) … }

S1 = { … P(S) … P(R) … V(S) execute service P(S) … V(R) … V(S) … }
S2 = { … P(S) … P(R) … V(S) execute service P(S) … V(R) … V(S) … }

Therefore, the priority ceiling of the semaphore *R* is equal to the task *R1* (because the semaphore *R* is used by the tasks *R1*, *S1* and *S2* and we know that the task *R1* is the highest priority) and the priority ceiling of the semaphore *S* is equal to the task *S1* (because the semaphore *S* is used by the tasks *S1* and *S2* and the priority task of *S1* is higher).

We suppose that the task *S2* is running when the task *S1* is created at the instant t3. We suppose also that the task *R1* is created at the instant t5.
In the Figure 2, a line in a high level indicates that the task is executing, a line in a low level indicates that the task is blocked or preempted by another task.

The following Table 2 explains more in details the example.

TABLE I. The event and its corresponding action in the Figure 2

Event	Action
t0	S2 begins execution.
t1	S2 locks S. The task S2 inherits the priority of S.
t2	The task S1 is created. As it has more priority than S2, it begins its execution.
t3	The task S1 fails to lock S as its priority is not higher than the priority ceiling of the locked S. The task S2 resumes the execution with the inherited priority of S.
t4	The task S2 locks R. The task S2 inherits the priority of R. The task R1 is created and preempts the (execution of S2 as it has the highest priority).
t5	The task R1 fails to lock R as its priority is not higher than the priority ceiling of the locked R. The task S2 resumes the execution of the critical section.
t6	The task S2 unlocks S.
t7	The task S2 executes a service.
t8	The task S2 locks S.
t9	The task S2 unlocks R and therefore has as priority the same as S. The task R1 becomes having the highest priority. As it has more priority than S2, it resumes its execution.
t10	The task R1 locks R.
t11	The task R1 executes the reconfiguration.
t12	The task R1 unlocks R.
t13	The task R1 terminates its execution.
t14	The task S2 unlocks S (thus S2 becomes having the lowest priority). Therefore, the task S1 resumes its execution.
t15	The task S1 locks S.
t16	The task S1 locks R.
t17	The task S1 unlocks S.
t18	The task S1 executes its service.
t19	The task S1 locks S.
t20	The task S1 unlocks R.
t21	The task S1 unlocks S.
t22	The task S1 achieves its execution.
t23	The task S2 resumes the execution and terminates its service.

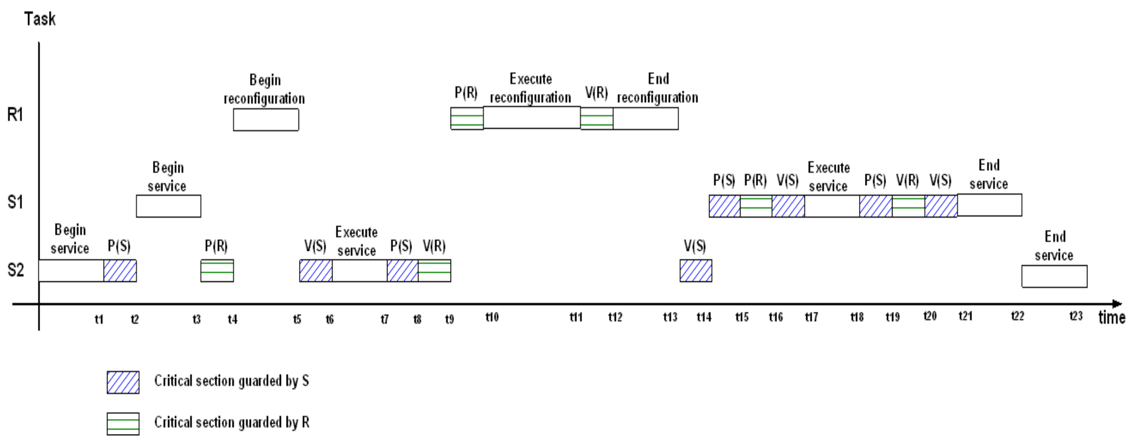

Task
R1
S1
S2
Begin reconfiguration
P(R)
Execute reconfiguration
V(R)
End reconfiguration
Begin service
P(S)
P(R)
V(S)
Execute service
V(R)
End service
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20 t21 t22 t23 time
Critical section guarded by S
Critical section guarded by R

Figure 2. The priority ceiling protocol applied to three tasks R1, S1 and S2

IV. SIMULATION WITH THE CHEDDAR TOOL

Cheddar, developed at the University of Brest, is a free real-time scheduling tool (http://beru.univ-brest.fr/~singhoff/cheddar). It is used to check temporal properties on real-time systems or applications. The tool provides a simulation engine and feasibility tests (Figure 3).

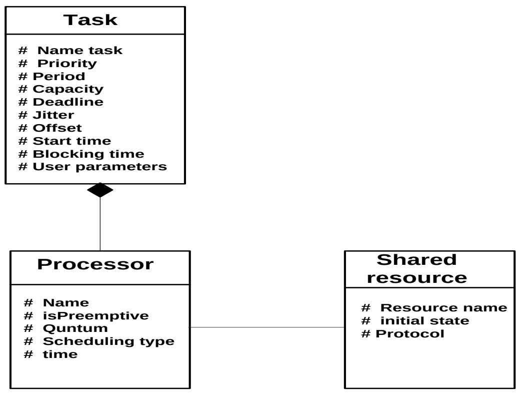


Figure 3. Cheddar Architecture

The use of Cheddar is very simple through its graphical interface.

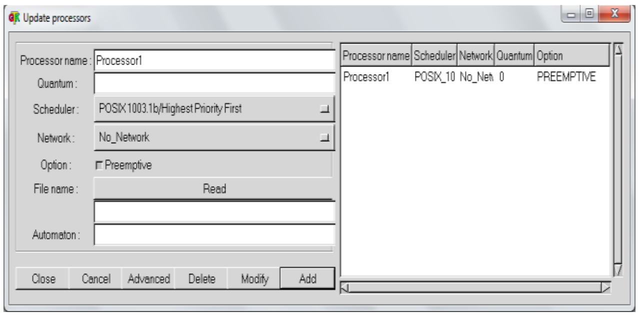
First of all, a processor must be defined via the Edit/Update processors. To do so, a name, a scheduler policy and the nature (it is preemptive or not) must be indicated.

Secondly, a memory address is precised in the Edit/Update address spaces by specifying its name.

Finally, all the tasks have to be defined in the Edit/Update tasks by indicating its characteristics.

Scheduling with Cheddar

- ✓ Definition of the processor: We aim to simulate the Priority Ceiling Protocol with the Cheddar tool. For that reason, we define a processor by giving him a name (*Processor1*), and choosing as scheduler "POSIX 1003.1b/Highest Priority First". We choose a preemptive scheduling which is RTEMS_PREEMPT (Figure 4).



Update processors

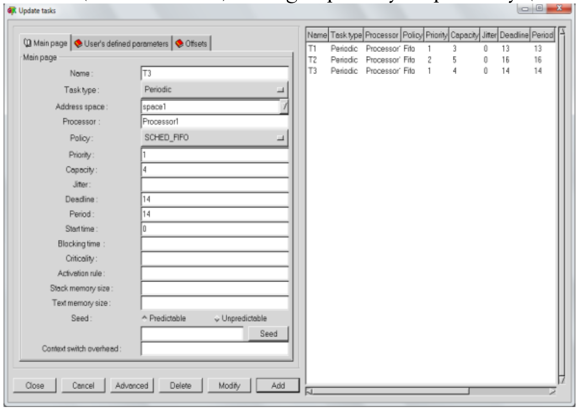
Processor name :	Processor1
Quantum :	
Scheduler :	POSIX 1003.1b/Highest Priority First
Network :	No_Network
Option :	Preemptive
File name :	Read
Automaton :	

Processor name	Scheduler	Network	Quantum	Option
Processor1	POSIX_10	No_Netv	0	PREEMPTIVE

Figure 4. Definition of a processor

We add also a space of addressing associated with the processor by indicating its name and the processor to which it is associated.

✓ Definition of the tasks: It is necessary to define the tasks and their properties. There are 4 types of tasks, periodic, aperiodic, fish or parametric. The kind of task which interests us here is periodic. After that, it is necessary to choose between policies SCHED_FIFO or SCHED_RR used for the scheduling of the same task priority. In this case, we choose SCHED_FIFO. It is also necessary to define the priority of the task. The priorities varies from 1 to 255, highest being 255. Lastly, it is necessary to specify work, starting times, deadline, and the period. These parameters are all obligatorily.
In our example, we create three tasks (*T1*, *T2*and *T3*) having as priority respectively (1, 2 and 1) (Figure 5).



Update tasks

Main page

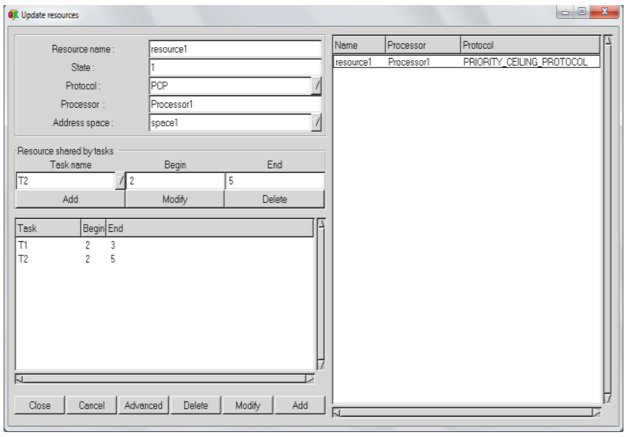
Name :	T3
Task type :	Periodic
Address space :	space1
Processor :	Processor1
Policy :	SCHED_FIFO
Priority :	1
Capacity :	4
Jitter :	
Deadline :	14
Period :	14
Start time :	0
Blocking time :	
Criticality :	
Activation rule :	
Stack memory size :	
Text memory size :	
Seed :	Predictable / Unpredictable
Context switch overhead :	

Name	Task type	Processor	Policy	Priority	Capacity	Jitter	Deadline	Period
T1	Periodic	Processor'	Fifo	1	3	0	13	13
T2	Periodic	Processor'	Fifo	2	5	0	16	16
T3	Periodic	Processor'	Fifo	1	4	0	14	14

Figure 5. Definition of a task with Cheddar

✓ Definition of the resource: As we need semaphores, resources should be added. We indicate the name of the resource, his initial value, and the intervals during which a task wants to reach the resource. These intervals cannot be null, i.e. a semaphore cannot be taken then delivered immediately. It is also necessary to specify the resource sharing protocol. Under Cheddar, the protocols available were No Protocol, PIP, PCP or IPCP.

In our example, we define a resource (*resource1*) associated with the PCP. The task *T1* needs to use this resource during the interval [2, 3] whereas the tasks *T2* wants to use it during the interval [2, 5] (Figure 6).



Update resources

Resource name :	resource1
State :	1
Protocol :	PCP
Processor :	Processor1
Address space :	space1

Resource shared by tasks

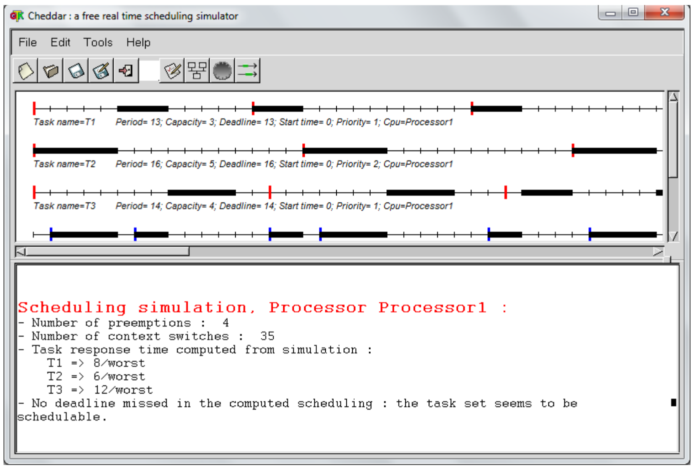
Task name	Begin	End
T2	2	5

Task	Begin	End
T1	2	3
T2	2	5

Name	Processor	Protocol
resource1	Processor1	PRIORITY_CEILING_PROTOCOL

Figure 6. Definition of a resource

The Figure 7 depicts an analysis of this test case by Cheddar. The top part of the window shows the thread scheduling. The worst case response time computed from this scheduling and with feasibility tests are shown in the bottom part of the window.


Cheddar : a free real time scheduling simulator
File
Edit
Tools
Help
Task name=T1
Period= 13; Capacity= 3; Deadline= 13; Start time= 0; Priority= 1; Cpu=Processor1
Task name=T2
Period= 16; Capacity= 5; Deadline= 16; Start time= 0; Priority= 2; Cpu=Processor1
Task name=T3
Period= 14; Capacity= 4; Deadline= 14; Start time= 0; Priority= 1; Cpu=Processor1

```
Scheduling simulation, Processor Processor1 :
- Number of preemptions :  4
- Number of context switches :  35
- Task response time computed from simulation :
    T1 => 8/worst
    T2 => 6/worst
    T3 => 12/worst
- No deadline missed in the computed scheduling : the task set seems to be
schedulable.
```

Figure 7. Scheduling with the Cheddar tool

V. CONCLUSION

This paper has as objective the study of Real-Time Task for Embedded Control System. We use the priority ceiling protocol as a method to ensure the scheduling between periodic tasks with precedence and mutual exclusion constraints.
The novelty of this paper is the study of dynamic reconfiguration with semaphore ensuring the following points: (i) blocking connections without blocking involved components; (ii) safety and correctness of the proposed solution; (iii) independence of any specific language; (iv) verification of consistency (i.e. logical constraints); (v) suitable for large-scale applications.

In the future work, we will be interested in the problem of allocation of real-time tasks to the devices of the execution environment and the assignment of these instances to the operating systems.

REFERENCES

[1] Merijn de Jonge, “Developing Product Lines with Third-Party Components”, Electronic Notes in Theoretical Computer Science, pages 63-80, 2009.

[2] David B. Stewart, Richard A. Volpe and Pradeep K. Khosla, “Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects”, IEEE Transactions on Software Engineering (23), pages 592-600, 1997.

[3] Roel Wuyts, Stephane Ducasse and Oscar Nierstrasz, “A data-centric approach to composing embedded, real-time software components”, The Journal of Systems and Software (74), pages 25-34, 2005.

[4] Ed Brinksma and all, “ROADMAP: Component-based Design and Integration Platforms”, http://www.artist-embedded.org, 2003.

[5] L. Sha, R. Rajkumar and J.P. Lehoczky, “Priority Inheritence Protocols: An Approach to Real-Time Synchronization”, IEEE Trans. on Computers. Vol. 39(9) (pp. 1175-1185)}, 1990.